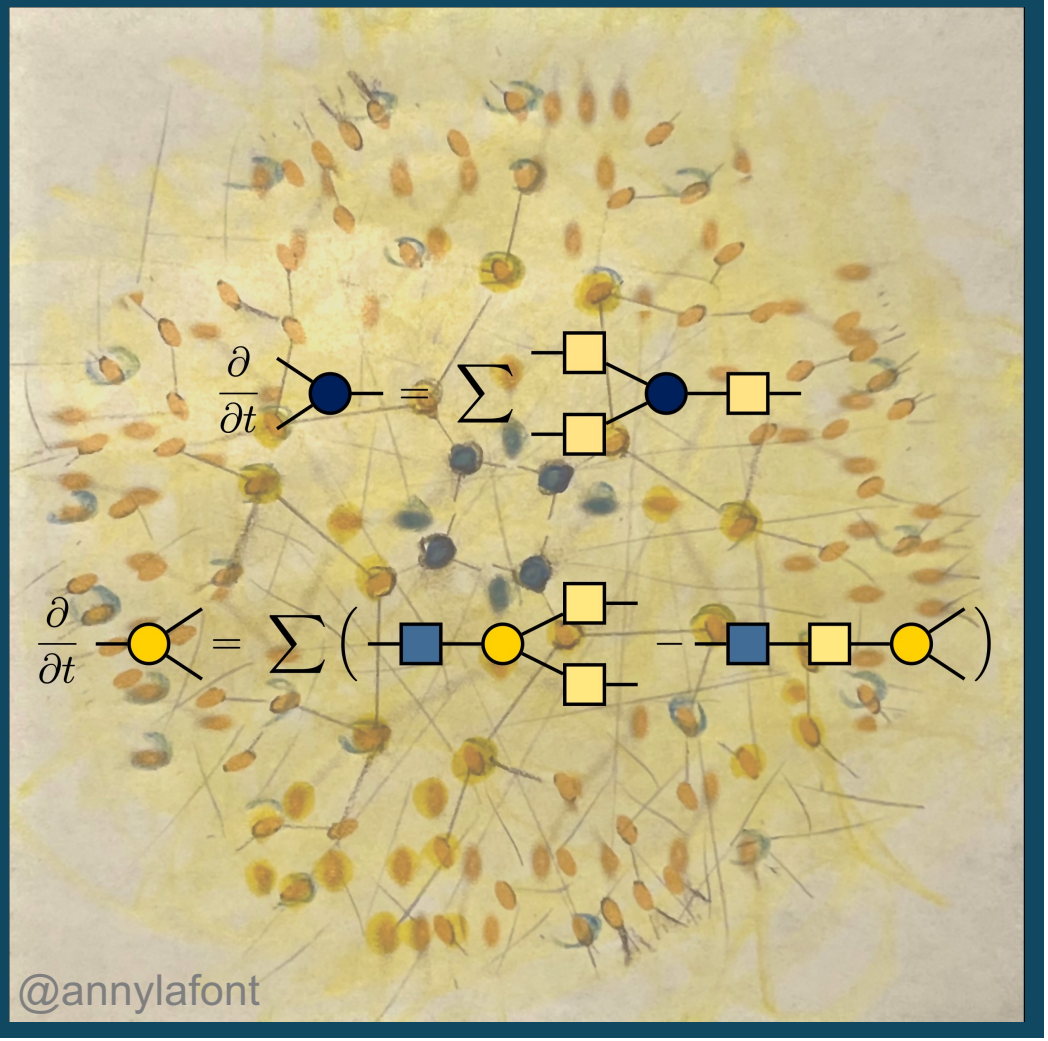
TENSO: Tree Tensor Network Equations for Numerically Exact Non-Markovian Open Quantum Dynamics

Ignacio Franco
University of Rochester



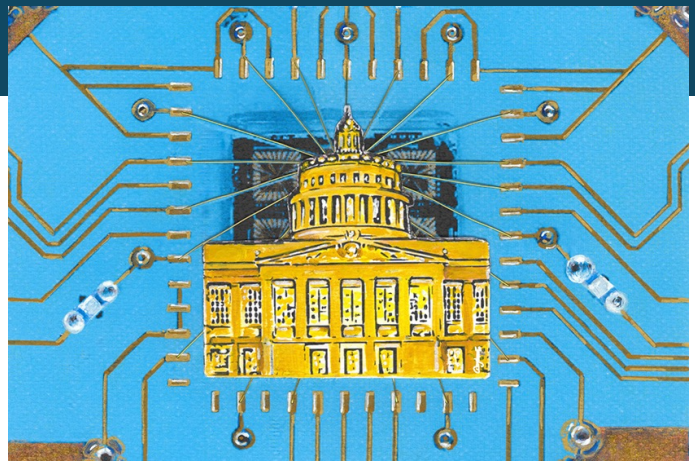
Thanks!

Xinxian Chen
Bexcitonic HEOM, TTN-HEOM, TENSO, Examples

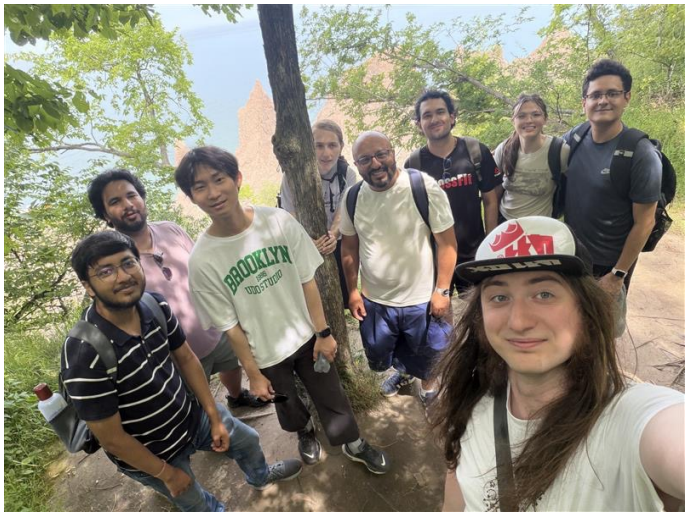
Juan Camilo Rodríguez, Michelle Anderson, Luchang Niu
Examples, Usage, Documentation

Developed this tutorial!

Thanks!



UR Center for Coherence and Quantum Science



DOE DE-SC0025334
NSF CHE-2416048
NSF CHE-2511834
NSF PHYS-2310657

University of Rochester

Structure of This Tutorial

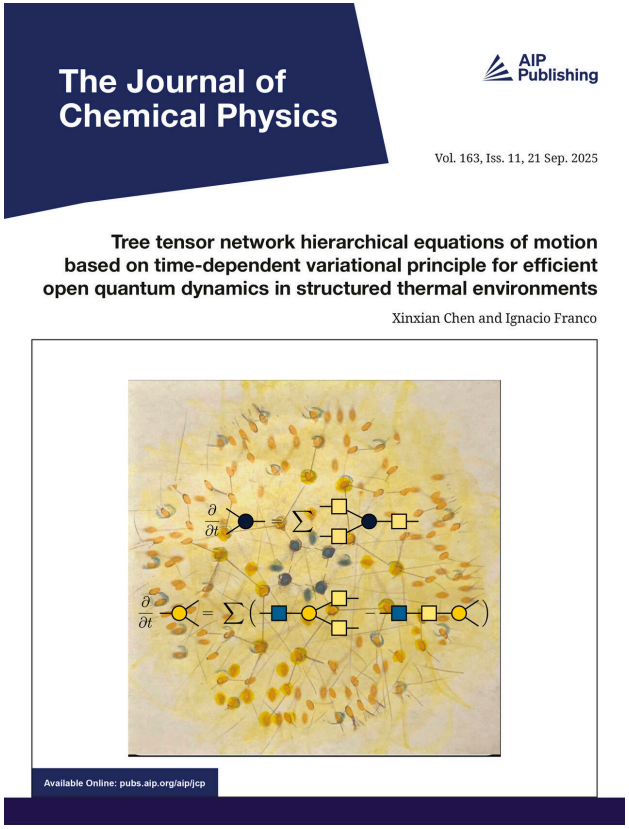
1. TENSO capabilities
2. Downloading and setting up TENSO
2. Numerical examples based on the “spin-boson” Hamiltonian
    - Thermal Relaxation of a Two-Surface Molecule
    - Tensor Trees versus Tensor Trains
    - Competing Thermal Environments
    - Laser-Driven Molecule Immersed in Solvent
    - Time-Propagation Strategies
3. Larger example: the Fenna-Matthews-Olson complex
4. Testing Convergence and Changing Units
5. Advanced TENSO features
6. Troubleshooting
7. Capstone Exercises: Superexchange and Entanglement Sudden Death

Tree Tensor Network HEOM and TENSO

https://github.com/ifgroup/pytenso

The TENSO package:

- Open Source
- Enables HEOM computations with highly structured environments by using tensor networks
- Implements tensor trains and tensor trees
- Has three propagation strategies (vmf, ps1, ps2) for robustness
- Allows for time-dependent driving and non-commuting fluctuations



TENSO: Usage Paper and Website

TENSO: Software Package for Numerically Exact Open Quantum Dynamics Based on Efficient Tree Tensor Network Decomposition of the Hierarchical Equations of Motion

Juan C. Rodriguez-Betancourt,[1] Michelle C. Anderson,[1] Luchang Niu,[2] Xinxian Chen,[1] and Ignacio Franco[1, 2, 3, a)]
[1)] *Department of Chemistry, University of Rochester, Rochester, New York 14627, United States*[b)]
[2)] *Department of Physics and Astronomy, University of Rochester, Rochester, New York 14627, United States*
[3)] *The Institute of Optics, University of Rochester, Rochester, New York 14627, United States*

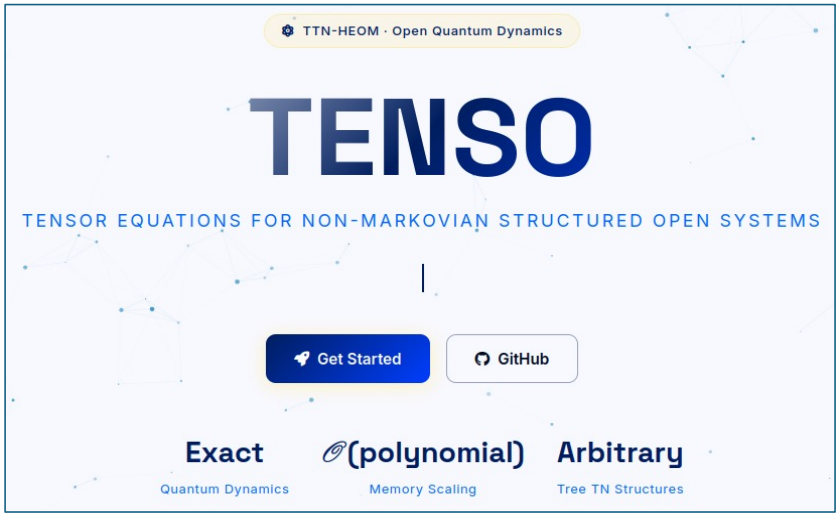
(Dated: 16 June 2026)

https://arxiv.org/pdf/2603.17711

In press in Journal of Chemical Theory and Computation

Covers this tutorial in detail

Discusses other uses and features of TENSO



https://ifgroup.github.io/pytenso/

Includes documentation, examples, and an input generator

TENSO: Environment Setup in CCR

We recommend using a package manager such as Anaconda to install TENSO and set up your environment

A. Import conda in CCR:

```
$ module use
/projects/academic/cyberwksp21/MODULES
$ module load libra_ava/devel
```

B. Restart the terminal:

```
$ source ~/.bashrc
```

C. Create TENSO conda environment:

```
$ conda create --name tenso python=3
matplotlib
$ conda activate tenso
```

D. Create a folder in your home directory:

```
$ mkdir tenso
$ cd ./tenso
```

TENSO: Environment Setup in CCR

E. Install TENSO:

```
$ git clone
https://github.com/ifgroup/pytenso.git
$ cd ./pytenso
$ python –m pip install –e .
```

F. Quick Sanity Check:

```
$ python -c ”from tenso.prototypes.heom import system_multibath;
print('ok')”
```

G. Register the Jupyter Kernel: This creates the kernel at /user/<username>/.local/share/jupyter/kernels/tenso/

```
$ conda activate tenso
$ python -m pip install ipykernel
$ python -m ipykernel install --user --name tenso --display-name
”Python (tenso)”
```

TENSO: Environment Setup in CCR

Now let's link TENSO with Jupyter, remember to replace <username> by your actual username

H. Edit your kernel.json file:

```
$ vi
/user/<username>/.local/share/jupyter/kernels/tenso/kernel.json
```

I. Replace what is inside by this:

```
{
 "argv": [
  "/user/<username>/.local/share/jupyter/kernels/tenso/launcher.sh",
  "-f",
  "{connection_file}"
 ],
 "display_name": "Python (tenso)",
 "language": "python",
 "metadata": {
  "debugger": true
 }
}
```

TENSO: Environment Setup in CCR

J. Create your launcher.sh:

```
$ vi /user/<username>/.local/share/jupyter/kernels/tenso/launcher.sh
```

K. The file is empty, add this:

```
#!/bin/bash
unset PYTHONPATH
unset PYTHONHOME
unset EBPYTHONPREFIXES
# Prevent user site leakage
export PYTHONNOUSERSITE=1
# Load module environment (provides conda)
module use /projects/academic/cyberwksp21/MODULES
module load libra_ava/devel
# ======================================================
# Activate the tenso conda environment
# ======================================================
source /projects/academic/cyberwksp21/SOFTWARE/Conda/etc/profile.d/conda.sh
conda activate tenso
# Launch kernel
exec /user/<username>/.conda/envs/tenso/bin/python \
     -m ipykernel_launcher "$@"
```

TENSO: Environment Setup in CCR

L. Make it executable:

```
$ chmod +x
/user/<username>/.local/share/jupyter/kernels/tenso/launcher.sh
```

Launch and Test:

1. Start the Jupyter app on ODD (no extra modules needed in the ODD form)
2. Select the “Python (tenso)” kernel
3. Test in a cell:

```
from tenso.prototypes.heom import system_multibath
```

Now clone the repo to get the examples:

```
$ git clone https://github.com/compchem-
cybertraining/Tutorials_TENSO.git
$ cd ./Tutorials_TENSO
```

TENSO: Environment Setup

We recommend using a package manager such as Anaconda to install TENSO and set up your environment

A. Download here:

1. Sign up
2. Sign in
3. Download

https://www.anaconda.com/download

B. Linux Install

4. Go to Downloads folder and open the terminal

```
$ bash ./anaconda_name.sh
```

5. Refresh the terminal

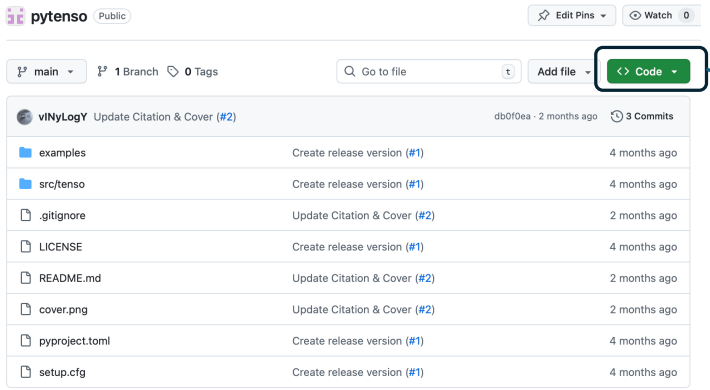
C. Create TENSO environment

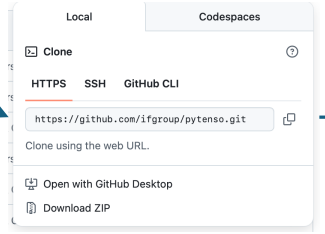
6. Set up the environment using the bash terminal:

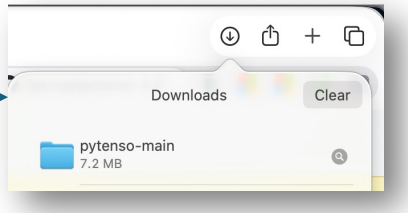
```
$ conda create --name tenso python=3.14 matplotlib
$ conda activate tenso
```

Installing TENSO

Download TENSO: https://github.com/ifgroup/pytenso







Graphical install

A. Move pytenso folder to home and unzip
B. Open the folder in the terminal
C. Install tenso in editable mode

```
$  python -m pip install -e .
```

Ubuntu install using bash terminal:

A. Install git

```
$ sudo apt install git
```

B. Clone the repository

```
$ git clone https://github.com/ifgroup/pytenso.git
```

C. Install tenso in editable mode

```
$ cd ./pytenso-main
$ python -m pip install -e .
```

Downloading Visual Studio Code

An IDE like Visual Studio is very useful

https://code.visualstudio.com

Ubuntu

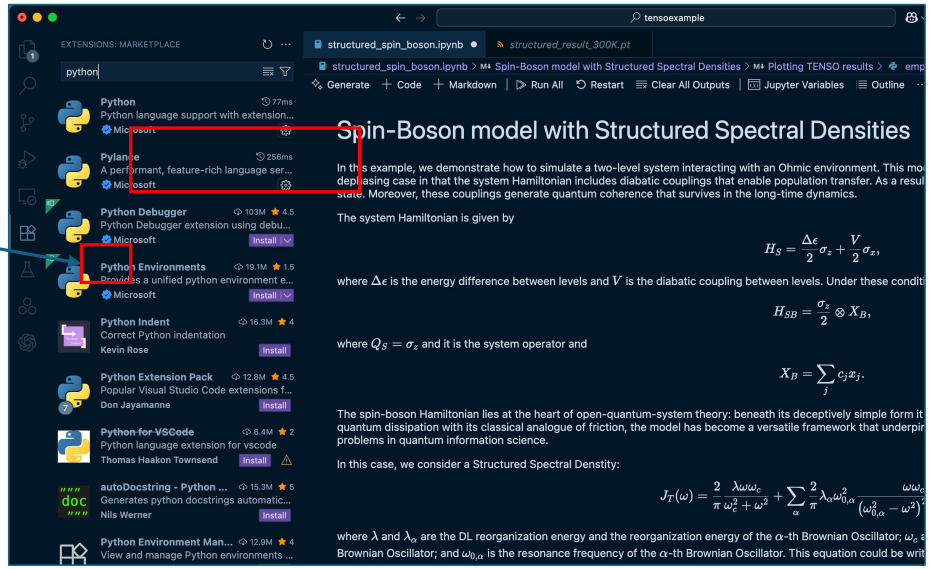
A. Download the .apt file from website
B. Install

```
$ sudo apt install ./code_1.107.1-1765982436_amd64.deb
```

C. Open VS Code

Setup Python and Jupyter

A. Go to the left panel and click on extensions
B. Search for python and install
C. Search for Jupyter and install



Create Jupyter Notebook (File… New File)
Select tenso environment or "kernel" on the top right

Setting up a TENSO Computation

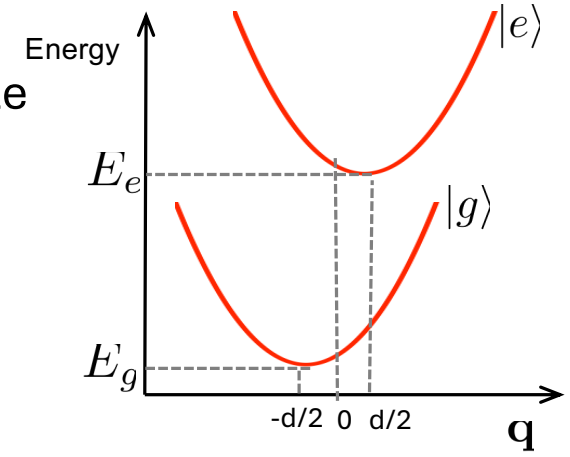
1. Import system libraries and TENSO modules
2. Use helper function `gen_bcf` to generate a bath correlation function from a spectral density
3. Define your system Hamiltonian and system-bath coupling
4. Pass your physical setup to `system_multibath` which will return a propagator object
5. Pass the propagator to the `tqdm` package which propagates the dynamics
6. Analyze the output density matrix dynamics

The Spin-Boson Problem

Two-level system (TLS) interacting with a thermal harmonic bath

Two electronic surface molecule **at finite temperature**

Other Uses:
- Nuclear Magnetic Resonance
- Spin impurities
- Qubits
- Useful benchmark



Vibrations/solvent

$$H_S = (E/2)\,\sigma_z + (V/2)\,\sigma_x$$

$$H_{\mathrm{B}} = \sum_{j=1}^{\infty} \omega_j a_j^\dagger a_j$$

$$H_{\mathrm{SB}} = Q_{\mathrm{S}} \otimes X_{\mathrm{B}}$$

$$X_{\mathrm{B}} = \sum_{j=1}^{\infty} (g_j a_j^\dagger + g_j^\star a_j)$$

$$Q_{\mathrm{S}} = \sigma_z \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Bath induces fluctuations in the energy levels

$$Q_{\mathrm{S}} = \sigma_x \qquad \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

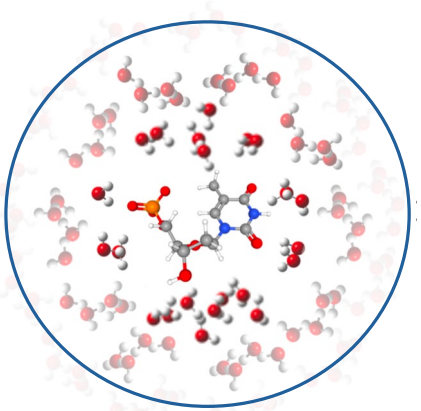
Bath induces fluctuations in the electronic couplings

Example 1: Thermal Relaxation of a Two-Surface Molecule

$$H_S = (E/2)\,\sigma_z + (V/2)\,\sigma_x \qquad \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

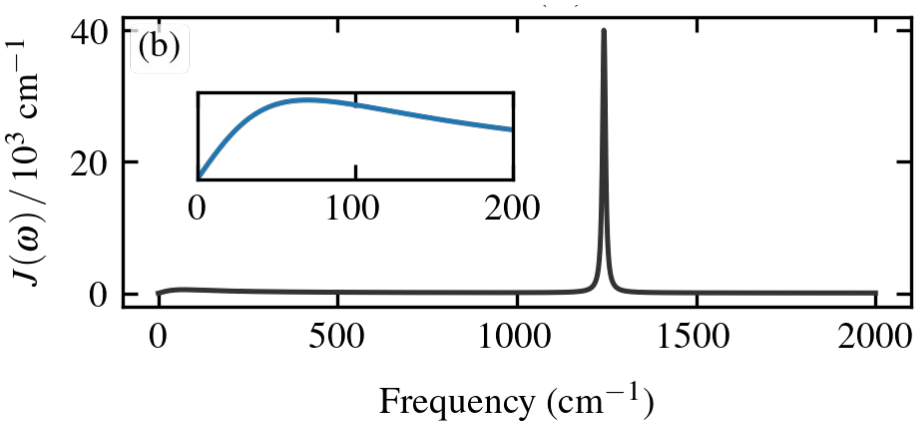
$$H_{\mathrm{SB}} = Q_{\mathrm{S}} \otimes X_{\mathrm{B}}$$

$$X_{\mathrm{B}} = \sum_{j=1}^{\infty} (g_j a_j^{\dagger} + g_j^{\star} a_j) \qquad Q_{\mathrm{S}} = \tfrac{1}{2}\sigma_z$$



Bath: Solvent (Drude-Lorentz) + one Vibration with finite lifetime (Brownian Oscillator)

$$J(\omega) = \sum_j |g_j|^2 \delta(\omega - \omega_j) \quad (\omega > 0)$$



Library Imports for TENSO Calculations

Input file: github.com/compchem-cybertraining/Tutorials_TENSO/tree/main/ex1_spin_boson_simple

Every TENSO calculation needs these libraries

```
from math import ceil
import os
import json as json
import numpy as np
from tqdm import tqdm

from tenso.prototypes.heom import system_multibath #Propagator helper function
from tenso.prototypes.bath import gen_bcf #Bath correlation function generator
import matplotlib.pyplot as plt
```

Test your TENSO installation:

Execute this with python to check the installation

Missing package error? Install it using package manager (Anaconda or Pip)

More confusing errors? Remove the TENSO environment (i.e. `conda env remove --name tenso`) and repeat the installation process

Generating the Bath Correlation Function

Function `gen_bcf` produces correlation functions as python objects

Needs reorganization energies, cut-off frequencies (DL) or widths (Brownian) and central frequencies (Brownian) as lists

```
bath_simulation = gen_bcf(
  re_d=[540], #Reorganization energy in cm-1
  width_d=[70], #Cutoff frequency of the DL spectral density in cm-1
  freq_b=[1243], # Central Frequency of the Brownian Spectral density in cm-1
  re_b=[161.6], #Reorganization energy of the Brownian Spectral density in cm-1
  width_b=[10], #Width of the spectral density in cm-1
  temperature=300, #Temperature in Kelvin
  decomposition_method='Pade', #Decomposition method for the bath correlation function
  n_ltc=1, #Number of low-temperature correction terms
)
```

You can add more features in the spectral densities by adding items to the lists: `re_d, width_d, freq_b, re_b, width_b`

You can choose the `'Pade'` or `'Matsubara'` decomposition. `'Pade'` is usually superior

Setting Up System Parameters

The `system_multibath()` helper function needs the Hamiltonian, coupling operators and bath correlation function as input and will return a propagator object

```
# Initialize system Hamiltonian
H = np.array([[1500/2, 600/2], [600/2, -1500/2]], dtype=np.complex128)


end_time = 1000.0 # End time in fs
dt = 1
wfn = np.array([1.0, 0.0], dtype=np.complex128)
out = 'example_1'

# Set up the propagator with given Hamiltonian and coupling
propagator = system_multibath(
        fname=out,
        init_rdo=np.outer(wfn, wfn.conj()),
        sys_ham=np.array([[1500/2, 600/2], [600/2, -1500/2]], dtype=np.complex128),
        sys_ops=[np.array([[0.5, 0.0], [0.0, -0.5]], dtype=np.complex128)],
        bath_correlations=[bath_simulation],
        dim=20,
        end_time=end_time,
    step_time=dt,
)
```

`dim`: HEOM depth
end_time: final propagation time; dt: time step
Lists are passed in to accommodate multiple baths and coupling operators

Running the Calculation

The package tqdm runs the propagator and gives a progress bar

```
# Run calculation
progress_bar = tqdm(propagator, total=ceil(end_time / dt))
for _t in (progress_bar):
    progress_bar.set_description(f'@{_t:.2f} fs')
```

This should take 4-10 minutes to run depending on your hardware

To summarize the main steps:

1. Import libraries
2. Generate the correlation function from the structured spectral densities
3. Generate a propagator for the given system setup including Hamiltonian, coupling, correlation function and tensor network options
4. Propagate dynamics

Information Printed During the Calculation

{'dim': 10, 'end_time': 1000.0, 'step_time': 1, 'save_checkpoint_to_file': True}
{'auxiliary_ps_method': 'ps2', 'auxiliary_step_time': None, 'cache_svd_info': True, 'dim': 10, 'dvr_length': 32, 'dvr_type': 'sine', 'end_time': 1000.0, 'frame_method': 'tree2', 'load_checkpoint_from_file': False, 'max_auxiliary_rank': 32, 'max_auxiliary_steps': None, 'metric': 're', 'ode_atol': 1e-07, 'ode_method': 'dopri5', 'ode_rtol': 1e-05, 'ps2_atol': 1e-07, 'ps2_ratio': 2.0, 'ps_method': 'vmf', 'rank': 3, 'renormalize': False, 'save_checkpoint_to_file': True, 'start_time': 0.0, 'step_time': 1, 'stepwise_method': 'mix', 'use_dvr': False, 'visualize_frame': False, 'vmf_atol': 1e-07, 'vmf_reg_method': 'extend', 'vmf_reg_type': 'ip'}
...

`dim`: depth of HEOM `rank`: rank of the tensor network for fixed rank methods only

`stepwise_method: mix` mixed propagation method.
In it `auxiliary_ps_method` (`ps2` in this case) runs first, then `ps_method` (`vmf` variable mean field/direct integration) runs

`frame_method: tree2` is a balanced binary tree

More technical information on integration tolerances, regularization, adaptive rank selection etc

Output Files

Output file: `'{out}.dat.log'` contains the density matrix in row major order

```
# time rdo00 rdo01 rdo10 rdo11
0.0 1.00000000+0.00000000j 0.00000000+0.00000000j 0.00000000+0.00000000j 0.00000000+0.00000000j
1.0 0.99683584-0.00000000j 0.00786785+0.05549515j 0.00786786-0.05549515j 0.00316438+0.00000000j
2.0 0.98767875+0.00000000j 0.03025228+0.10513464j 0.03025228-0.10513464j 0.01232146+0.00000000j
3.0 0.97347391+0.00000000j 0.06384260+0.14415478j 0.06384260-0.14415479j 0.02652630-0.00000000j
4.0 0.95560609+0.00000000j 0.10398967+0.16963628j 0.10398967-0.16963629j 0.04439413-0.00000000j
```

Debug file: `'{out}.debug.log'` info on the structure of the tensor network, i.e. largest rank, how hard the integrator had to work on each step (`ode_steps`)

```
# time ode_steps max_rank tr norm
# # ([H]0)-([H]3) ([H]3)-([H]1) ([H]3)-([H]2)
0.0 0 3 1.0 1.0
# 3 3 3
1.0 140 16 1.0000002181307674 1.0036323725262248
# 4 16 16
2.0 140 18 1.0000002181307592 1.0142676340396264
```

Checkpoint file: `'{out}.pt'` if requested by a `'save_checkpoint_to_file=True'` during propagator setup.

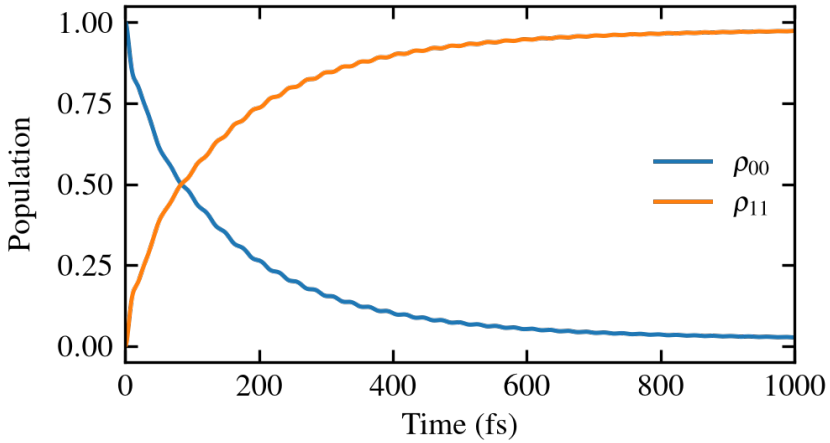
This binary file contains information needed to restart an interrupted calculation and is loaded by setting `'load_checkpoint_from_file=True'` in the propagator setup.

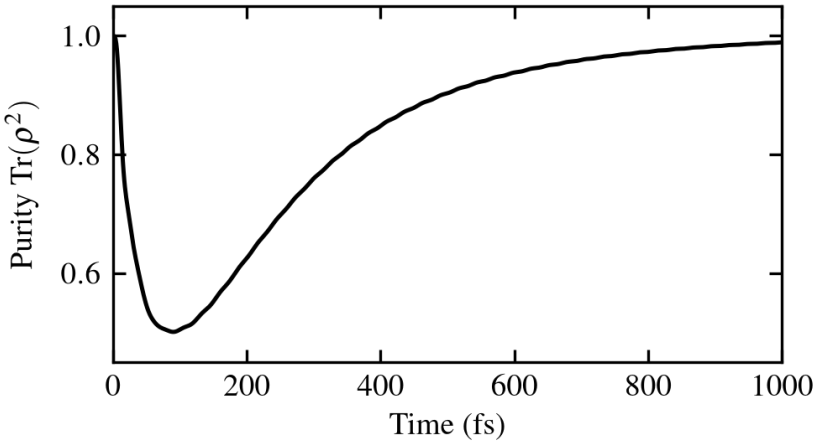
Results of Thermal Relaxation

Use this to plot or download the plotting script

```
out = 'example1'
# Load results from the output file into a local array
results = np.loadtxt(out+".dat.log",dtype=np.complex128)
# Plot the populations of the two state of the spin-boson problem
plt.plot(np.real(results[:,0]),np.real(results[:,1]),np.real(results[:,0]),np.real(results[:,4]))
plt.xlabel("Time (fs)")
plt.ylabel("Population")
plt.savefig(out+'.png')
plt.show()
```

github.com/compchem-cybertraining/Tutorials_TENSO/blob/main/ex1_spin_boson_simple/example1_plot.py





Note: Purity is a basis-independent measure of how pure the state is (describable by a wavefunction).

Exercise 1: Thermal Relaxation of a Molecule

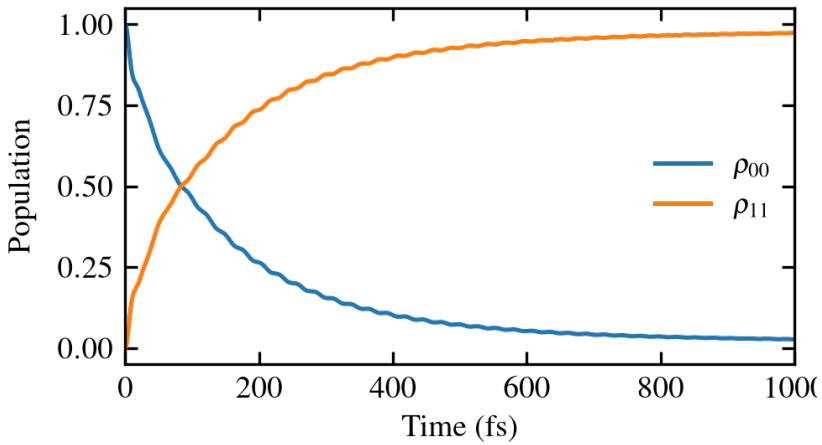
- **Run it:** Launch `example1.py`, then `example1_plot.py` starting in the excited state. How long does it take to reach thermal equilibrium?
- **Raise Temperature** in `gen_bcf` from **300 K→ 600 K** . How does the dynamics and thermal equilibrium change?
- **Simplify the bath:** Remove the intramolecular vibration. What impact does this have on dynamics and computational speed?
- **Weaker coupling:** reduce `re_d` from `[540]` to `[54]` (10× weaker) . What is the impact on dynamics? Are oscillations enhanced or suppressed?

Results

Use this to plot or download the plotting script

```
out = 'example1'
# Load results from the output file into a local array
results = np.loadtxt(out+".dat.log",dtype=np.complex128)
# Plot the populations of the two state of the spin-boson problem
plt.plot(np.real(results[:,0]),np.real(results[:,1]),np.real(results[:,0]),np.real(results[:,4]))
plt.xlabel("Time (fs)")
plt.ylabel("Population")
plt.savefig(out+'.png')
plt.show()
```

github.com/compchem-cybertraining/Tutorials_TENSO/blob/main/ex1_spin_boson_simple/example1_plot.py



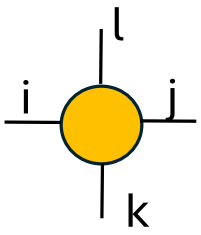
As temperature increases, excited state population increases

Wiggles in the dynamics are from the Brownian peak. Computation is much faster without the Brownian's features.

Weaker coupling leads to sustained coherent oscillations.

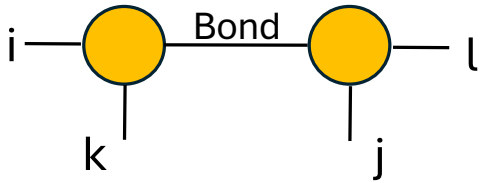
Tree Structures: Tensor Train and Trees

Tensor network is formed by repeated singular value decompositions of high-order tensor



4D Tensor

SVD → 3D Tensors

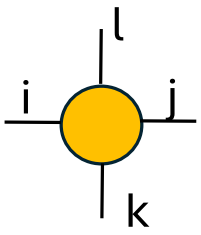


Bond dimension or rank: number of singular values retained *R*

The tensor network structure can impact the stability and efficiency of the calculation

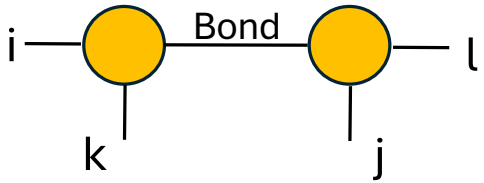
Tree Structures: Tensor Train

Tensor network is formed by repeated singular value decompositions of high-order tensor



4D Tensor

SVD → 3D Tensors



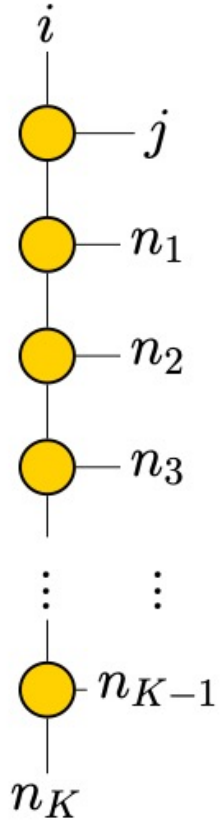
Bond dimension or rank: number of singular values retained *R*

TENSO admits arbitrary TN structure

It automatically constructs:
- Balanced tensor tree
- Tensor Train or Matrix Product State

Tensor Train (TT)
- Variable distance of bath degrees of freedom to the system dofs

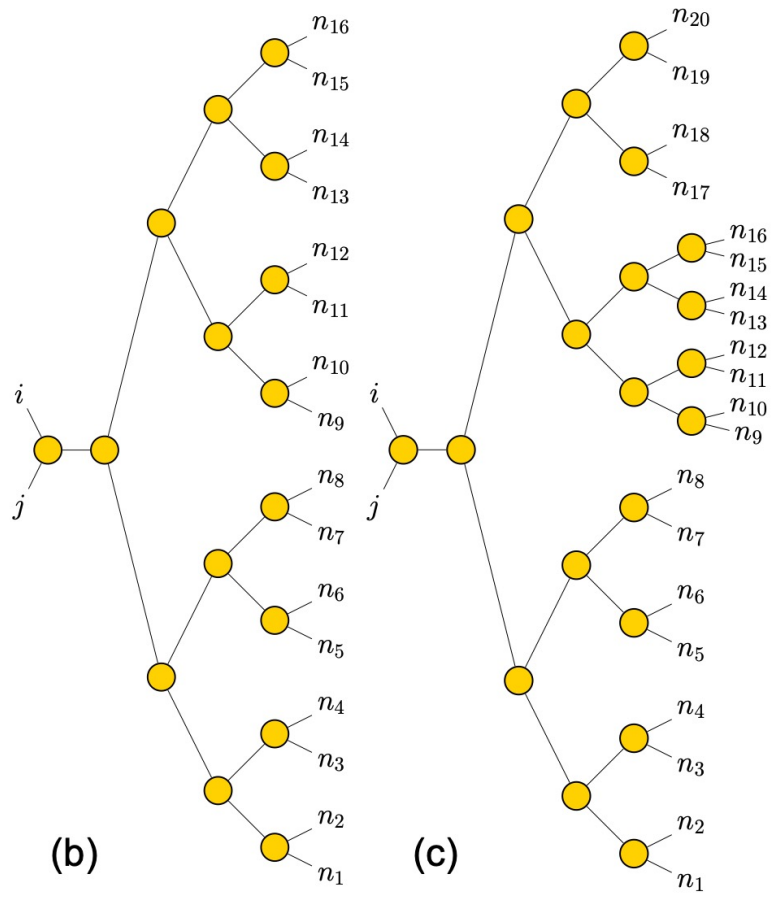
- Useful if a bath component interact more strongly with the system than others



Tree Structures: Balanced Binary Tree

BTT: distance between the root tensor and the leaves is minimized

Best choice if nothing is known about the open quantum system

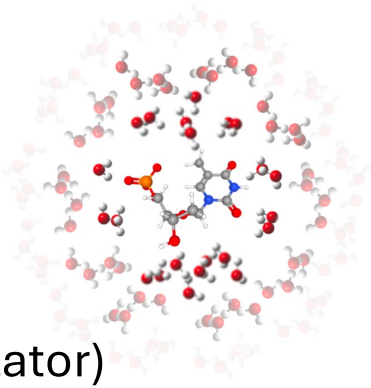


Example 2: Tensor Trees Versus Train

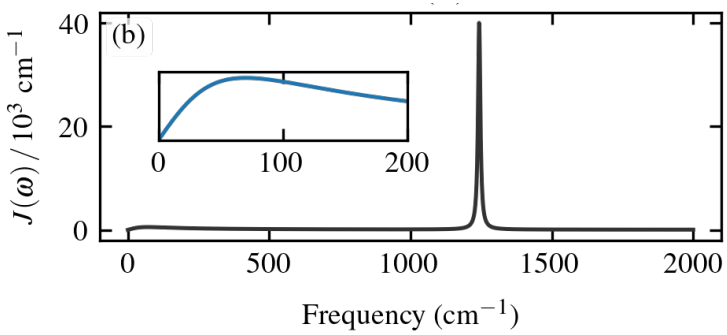
Same two-surface molecule

$$H_S = (E/2)\,\sigma_z + (V/2)\,\sigma_x$$

$$Q_S = \tfrac{1}{2}\sigma_z$$



Solvent (Drude-Lorentz) + 1 Intramolecular Vibration (Brownian Oscillator)



How does the choice of tensor network influence the dynamics?

Setting Up the Comparison

Input file: github.com/compchem-cybertraining/Tutorials_TENSO/tree/main/ex2_spin_boson

Import libraries

Define the name of the outputs and the `'frame_method'`. We use balanced tree as `'tree2'` and tensor train as `'train'`.

```
out_tree = 'tree2' #Output name
out_name = 'train' #Output name
outs = {
'tree2': out_tree,
'train': out_name
}
```

Define the BCF:

```
bath_simulation = gen_bcf(
  re_d=[540], #Reorganization energy in cm-1
  width_d=[70], #Cutoff frequency of the DL spectral density in cm-1
  freq_b=[1243], # Central Frequency of the Brownian Spectral density in cm-1
  re_b=[161.6], #Reorganization energy of the Brownian Spectral density in cm-1
  width_b=[10], #Width of the spectral density in cm-1
  temperature=300, #Temperature in Kelvin
  decomposition_method='Pade', #Decomposition method for the bath correlation function
  n_ltc=1, #Number of low-temperature correction terms
)
```

Propagate the Dynamics

Define system dynamics along with `'end_time'` and `'step_time'`:

```
H = np.array([[1500/2, 600/2], [600/2, -1500/2]], dtype=np.complex128)

end_time = 1000.0 #fs
dt = 1
wfn = np.array([1.0, 0.0], dtype=np.complex128)
```

Propagate adding the `'frame_method'` parameter and setting it to `'tree2'` or `'train'` in turn using a *for* loop

```
for out in outs:
        propagator = system_multibath(
                fname=out,
                init_rdo=np.outer(wfn, wfn.conj()),
                sys_ham=np.array([[1500/2, 600/2], [600/2, -1500/2]],
        dtype=np.complex128),
                sys_ops=[np.array([[0.5, 0.0], [0.0, -0.5]], dtype=np.complex128)],
                bath_correlations=[bath_simulation],
                dim=25,
                end_time=end_time,
                step_time=dt,
                frame_method=out,
                )
        progress_bar = tqdm(propagator, total=ceil(end_time / dt))
        for _t in (progress_bar):
                progress_bar.set_description(f'@{_t:.2f} fs')
```

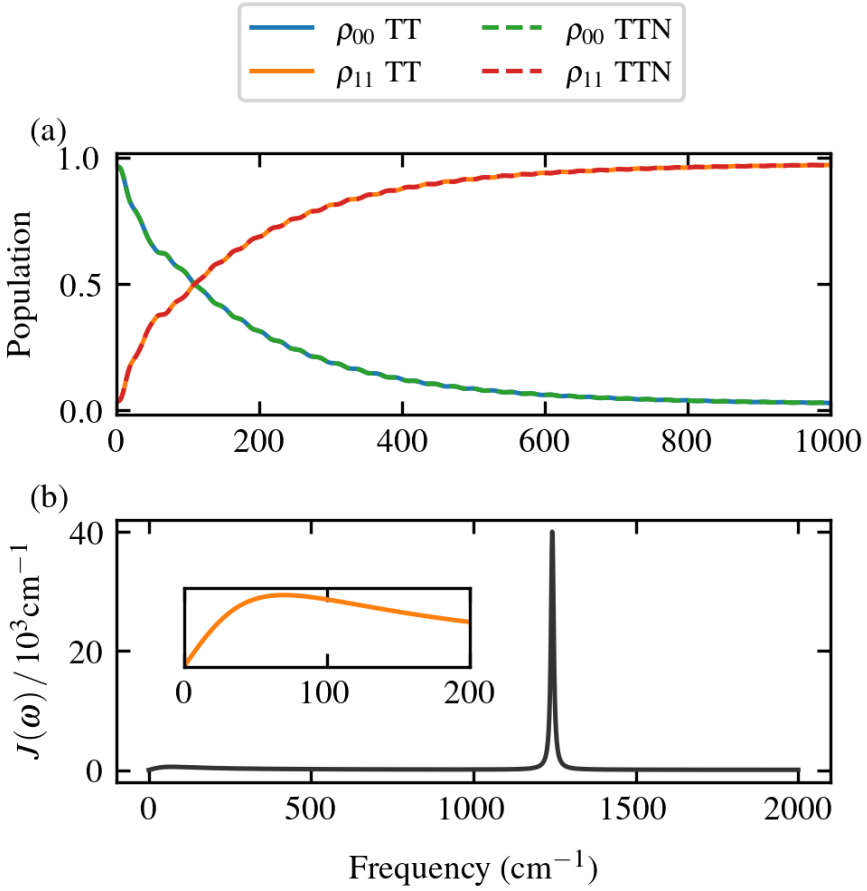
Exercise 2: Influence of Tensor Network Structure

- **Converged answers:** run the example, `example2.py, example2_plot.py`. Do you see any difference between the results given from the tree and train network? Should you?

- **Computational speed:** the **tqdm** bar reports each run's speed (iterations/s). Which tensor network structure is faster? How large is the difference?

- **A third structure:** add `'naive':'naive'` to `outs.` You will also have to add the line `stepwise_method='simple'` to the propagator. In this case, the tensor will not be factorized into a tree structure. How long would it take to run this system with this method?

Results



Plotting script: `example2_plot.py`

If the dynamics are converged the tensor network structure should not matter

Convergence with rank, and speed of computation, depend on the tensor network

Using the naïve framework is a poor choice. It takes hours (this is the standard HEOM)!

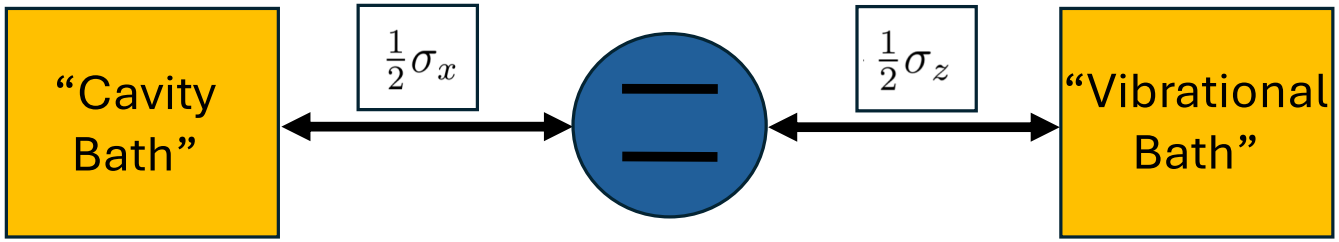
Example 3: Competing Thermal Environments

Spin-boson coupled to two identical baths via non-commuting operators:

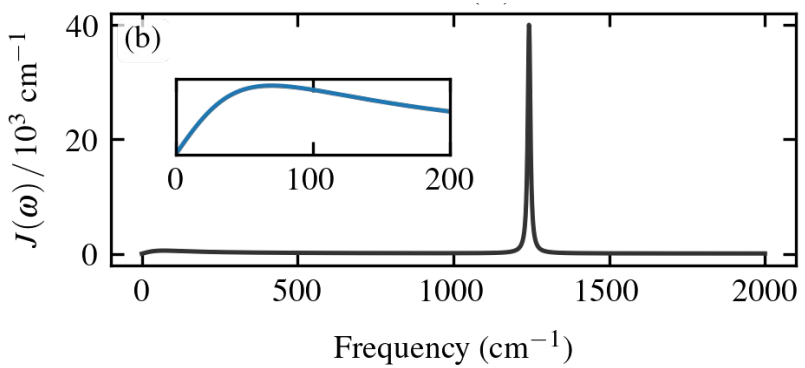
$$H_{\mathrm{SB}} = \tfrac{1}{2}\sigma_x \otimes X_{\mathrm{B}}^{(1)} + \tfrac{1}{2}\sigma_z \otimes X_{\mathrm{B}}^{(2)}$$

Non-commuting fluctuations lead to complex dynamics

Example: Molecule in Cavity



Environment: 1 Drude-Lorentz + 1 Brownian Oscillator



Hard for QuAPI!

Setting Up the Bath Correlation Function

Input file: github.com/compchem-cybertraining/Tutorials_TENSO/tree/main/ex3_spin_boson_noncommuting_fluctuations

Import libraries

Keep the structured bath correlation function. The output file is "`two_baths`"

```
# Generate the bath correlation function and decompose
# it for HEOM propagation
bath_simulation = gen_bcf(
        re_d=[540],
        width_d=[70],
        freq_b=[1243],
        re_b=[161.6],
        width_b=[10],
        temperature=300,
        decomposition_method='Pade',
        n_ltc=1,)

end_time = 500 # Propagation time in fs
dt = 1 # Time step size in fs
wfn = np.array([0.0, 1.0], dtype=np.complex128) # Initial wavefunction
out="two_baths" # Base name for the output
```

Setting Up Coupling Operators and Propagator

Define two new bath coupling operators:

```
bath_op1 = np.array([[-0.5,0.0],[0.0,0.5]],dtype=np.complex128)
bath_op2 = np.array([[0.0,0.5],[0.5,0.0]],dtype=np.complex128)
```

Prepare the propagator. Same Hamiltonian + two bath operators + two bath correlation functions (identical in this case).

```
# Generate the propagator that will advance the system
propagator = system_multibath(
        fname=out, # Specify output file
        init_rdo=np.outer(wfn, wfn.conj()), # Set system initial condition
        sys_ham=np.array([[-750.0,300.0],[300.0,750.0]], dtype=np.complex128), # Set the
system Hamiltonian (cm^{-1})
        sys_ops=[bath_op1,bath_op2], # Set the H_{SB} system-bath coupling operator
(cm^{-1})
        bath_correlations=[bath_simulation, bath_simulation], # Specify a bath
correlation function generated by gen_bcf
        dim=30, # Hierarchy depth
        end_time=end_time, # End time of the simulation
        step_time=dt,) # Time step size

# Perform the dynamics simulation and generate a progress bar
progress_bar = tqdm(propagator,total=ceil(end_time/dt))
for _t in progress_bar:
        progress_bar.set_description(f'@{_t: .2f} fs')z
```

Run the propagator

Exercise 3: Two Competing Environments

- **Run the examples:** run `example3.py, example3_plot.py`. Set your `end_time = 200` as this is a longer calculation

- **One bath at a time:** run with coupling given by only **bath_op1**, then only **bath_op2**. Compare the population and coherence dynamics.

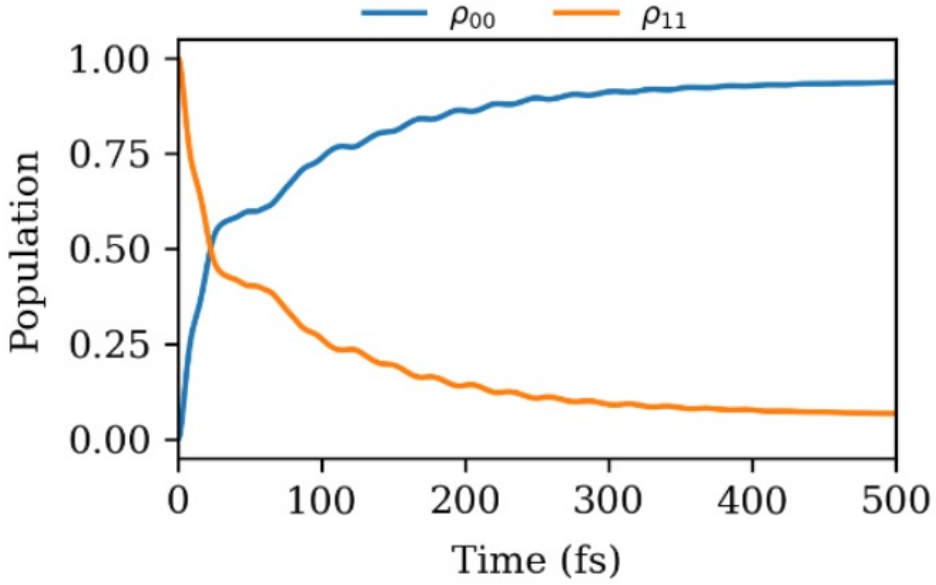
- **Coherence decay:** set V to 0 and initialize in an equal-amplitude superposition state
  - Couple only along σ_z.
  - Replace with a mixture of σ_z and σ_x coupling ($c\sigma_x + d\sigma_z$ with c+d=1) and run for 100 fs
  - How does the decay of coherence change as the non-commuting fluctuation is introduced?

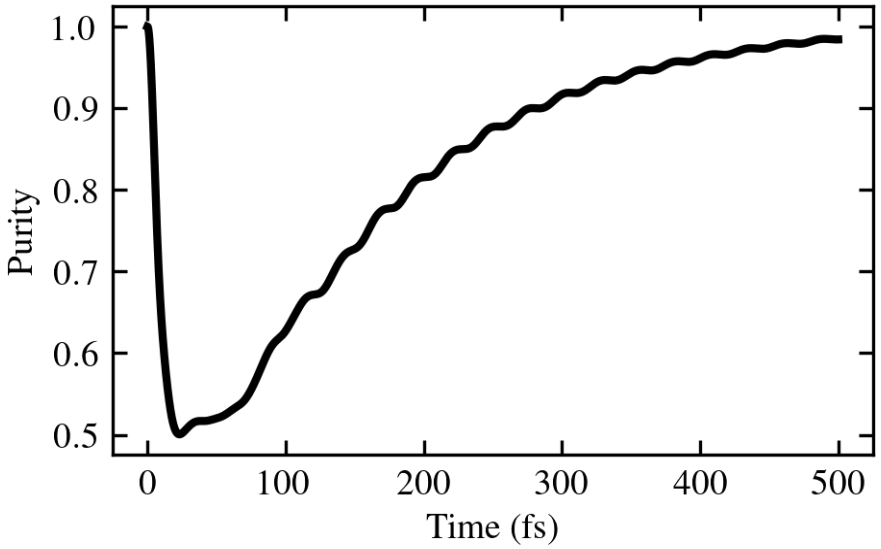
Results: Complex Population Dynamics

Single bath coupled to σ_x yields larger population oscillations and slower convergence towards thermal equilibrium

Pure dephasing (V=0 and σ_z coupling) leads to fast coherence loss but no relaxation.

Non-commuting fluctuates lead to more complicated and longer-lived coherence dynamics



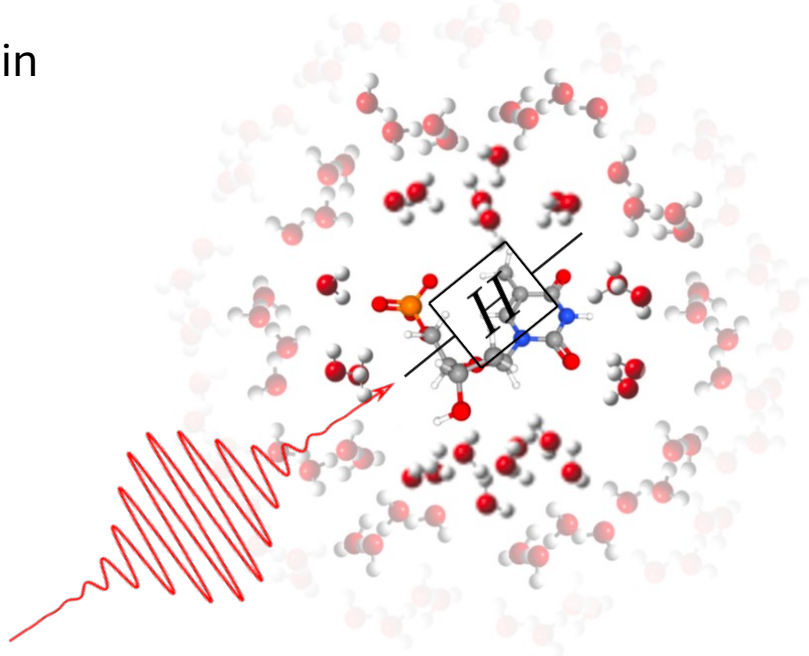


Example 4: Laser-Driven Molecule in Solvent

Light-Matter interactions introduces time-dependence in Hamiltonian.

Needed for:

- Photophysics and photochemistry
- Spectroscopy
- Quantum control
- Quantum information



Laser pulse is designed to execute a Hadamard gate:

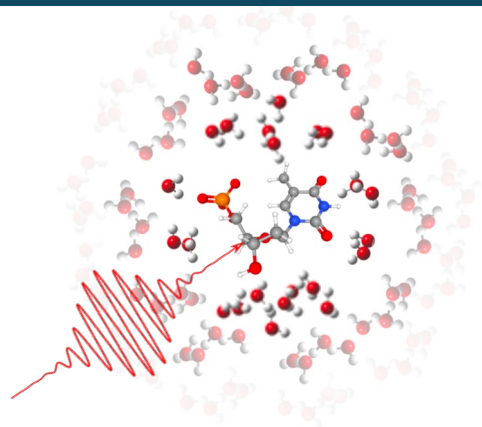
$$|\Psi\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow |\Psi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Example 4: Laser-Driven Molecule in Solvent

Driven molecule

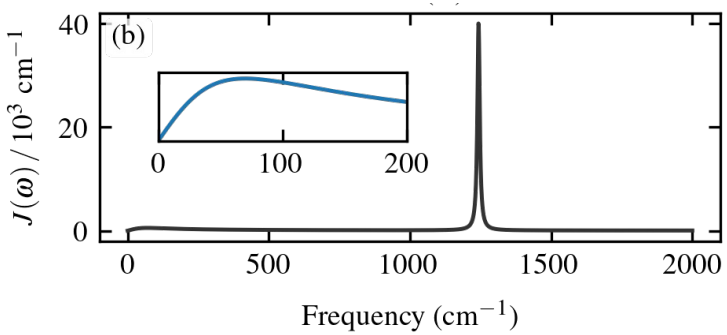
$$H_{\mathrm{S}} = \sigma_z \frac{E}{2} - \sigma_x \mu_0 E_0(t) \cos(\omega t)$$

$$\Omega(t) = -\mu_0 E_0(t)$$



System-Bath Coupling $H_{\mathrm{SB}} = Q_{\mathrm{S}} \otimes X_{\mathrm{B}}$ $Q_{\mathrm{S}} = \frac{1}{2}\sigma_z$

Solvent (1 Drude-Lorentz) + 1vibration (Brownian Oscillator)



Setting Up the Bath Correlation Function

Input file: https://github.com/compchem-cybertraining/Tutorials_TENSO/tree/main/ex4_spin_boson_time_dependent

Generate the BCF

```
bath_simulation = gen_bcf(
  re_d=[540], #Reorganization energy in cm-1
  width_d=[70], #Cutoff frequency of the DL spectral density in cm-1
  freq_b=[1243], # Central Frequency of the Brownian Spectral density in cm-1
  re_b=[161.6], #Reorganization energy of the Brownian Spectral density in cm-1
  width_b=[10], #Width of the spectral density in cm-1
  temperature=300, #Temperature in Kelvin
  decomposition_method='Pade', #Decomposition method for the bath correlation
function
  n_ltc=1, #Number of low-temperature correction terms
)
```

Define the dynamics parameters and initial state. Use smaller timestep

```
end_time = 40.0 # fs
dt = 0.05 # fs
wfn = np.array([0.0, 1.0], dtype=np.complex128) # Start in Ground State |0>
(convention [exc, gnd])
```

Setting Up the Time-Dependent Drive

Define the drive parameters and drive function using TENSO's default units of wavenumbers and femtoseconds

```
t0_fs = 6.0
omega_rad_fs = 0.2825477
# The dipole has been incorporated into this value
interaction_max_cm = 3133.625
sigma_fs = 2.1233045
# Laser field function
def laser_field_hadamard(t):
      #Returns the laser field interaction energy at time t (fs)
      envelope = np.exp(-0.5 * ((t - t0_fs) / sigma_fs)**2)
      # Unitless argument inside cos: (rad/fs) * fs = radians
      carrier = np.cos(omega_rad_fs * t)
      return -interaction_max_cm * envelope * carrier
```

Setting Up the Propagator

The needed `'td_f'` and `'td_op'` are added. `'td_f'` is the time-dependent field and `'td_op'` is the coupling operator with the field.

```
out = 'laser_example'
propagator = system_multibath(
        fname='out',
        init_rdo=np.outer(wfn, wfn.conj()),
        # System Hamiltonian
        sys_ham=np.array([[750, 0.0], [0.0, -750]],
        dtype=np.complex128),
        # System-Bath coupling operator (Sigma_z / 2)
        sys_ops=[np.array([[0.5, 0.0], [0.0, -0.5]],
        dtype=np.complex128)],
        bath_correlations=[bath_simulation],
        # Time-dependent driving function & Operator
        td_f=laser_field_hadamard,
        td_op=np.array([[0.0, 1.0], [1.0, 0.0]],
        dtype=np.complex128), #(sigma_x)
        dim=30,
        end_time=end_time,
        step_time=dt,
)
```

Exercise 4: Laser-Driven Molecule in Solvent

- **Does the gate operation perform as expected?** Run **example4.py**; the simulated laser pulse hits at t ≈ 6 fs and, ideally, should result in the execution of a Hadamard gate.

Are the final populations consistent with a successful Hadamard?

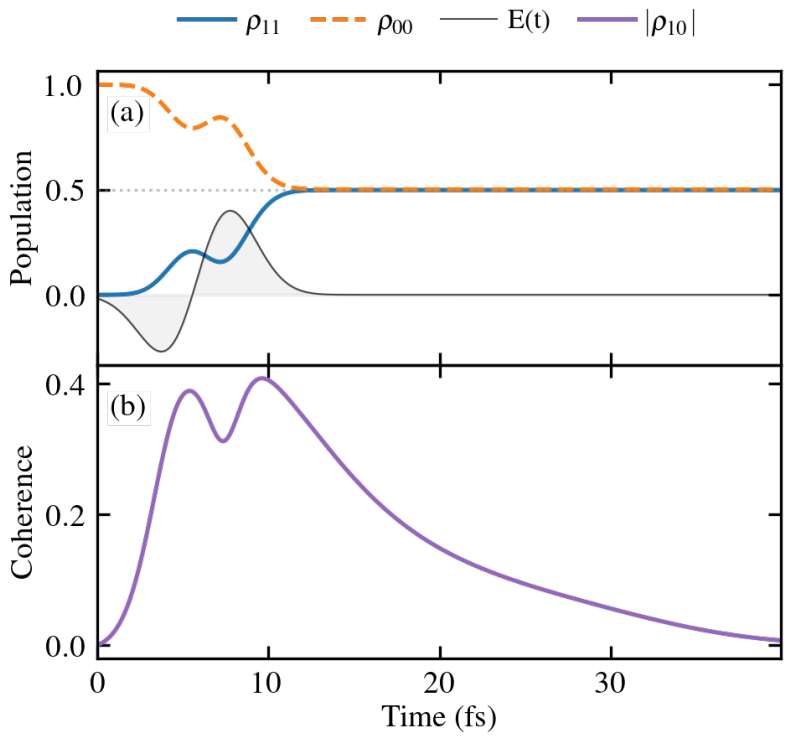
What about the coherences?

- **Modifying environmental coupling strength:** Double the coupling of the DL bath to `re_d = [1080]`. Then try halving it. How does this impact the dynamics?

Does modifying the strength of the Brownian term only have larger or smaller impact?

Results

Noisy Hadamard gate does not reach the maximally coherent state due to decoherence.



Plotting script: `example3_plot.py`

Coherence decays more quickly when the coupling to the bath is stronger.

In this case, changing the DL term has a larger impact than changing the Brownian term.

Propagation Schemes in TENSO

Projector Splitting 1

- Fixed rank one-site algorithm
- Trotterization algorithm
- Errors determined by time step size
- Challenging to parallelize

Projector Splitting 2

- Adaptive rank
- Trotterization algorithm
- Errors determined by time step size
- Two-site algorithm
- Rank grows fast

More details in: J. Chem. Phys. 163, 104109 (2025)

Direct Propagation of Master Equation for Core Tensors (in TENSO: vmf)

- Admits high-order numerical propagation schemes (e.g. Runge-Kutta)
- Fixed rank one-site algorithm
- Requires regularization
- Challenging for early times

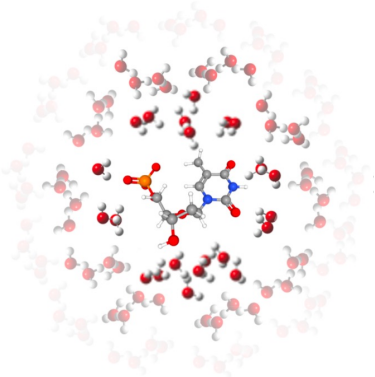
Default: Propagates with ps2 at early times then switches to vmf

Example 5: Time-Propagation Strategies

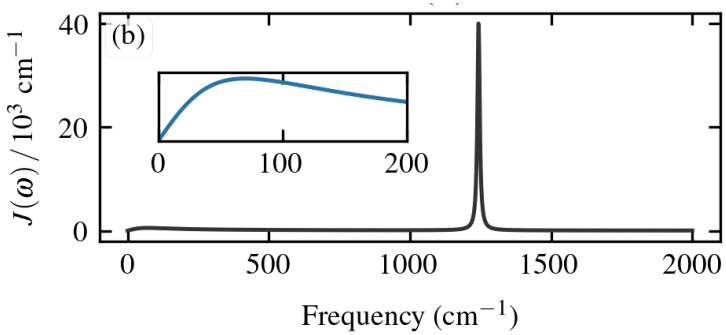
Same physical system

$$H_S = (E/2)\,\sigma_z + (V/2)\,\sigma_x$$

$$H_{\mathrm{SB}} = Q_{\mathrm{S}} \otimes X_{\mathrm{B}} \qquad Q_{\mathrm{S}} = \tfrac{1}{2}\sigma_z$$



Spectral Density:



We will repeat our computations with different propagation strategies

Setting Up a Comparison of Propagation Schemes

Input file: https://github.com/compchem-cybertraining/Tutorials_TENSO/tree/main/ex5_spin_boson_propagation

Import the libraries as usual

Define the name of the outputs and the propagation scheme `'ps_method'`:

```
propagation_1 = 'ps1' #Output name
propagation_2 = 'vmf' #Output name
outs = {
    'ps1': propagation_1,
    'vmf': propagation_2
}
```

Define the same BCF as before

```
bath_simulation = gen_bcf(
  re_d=[540], #Reorganization energy in cm-1
  width_d=[70], #Cutoff frequency of the DL spectral density in cm-1
  freq_b=[1243], # Central Frequency of the Brownian Spectral density in cm-1
  re_b=[161.6], #Reorganization energy of the Brownian Spectral density in cm-1
  width_b=[10], #Width of the spectral density in cm-1
  temperature=300, #Temperature in Kelvin
  decomposition_method='Pade', #Decomposition method for the bath correlation function
  n_ltc=1, #Number of low-temperature correction terms
)
```

Loop Through the Different Propagation Schemes

Propagate as usual but specify that only one propagation scheme should be used for the entire calculation

In this example, `'ps1'` and `'vmf'` are used. Since these methods are fixed rank, we need to specify the rank

```
for out in outs:
  propagator = spin_boson(
    fname=out,
    init_rdo=np.outer(wfn, wfn.conj()),
    sys_ham=np.array([[1500/2, 600/2], [600/2, -1500/2]], dtype=np.complex128),
    sys_op=np.array([[0.5, 0.0], [0.0, -0.5]], dtype=np.complex128),
    bath_correlation=bath_simulation,
    dim=20,
    end_time=end_time,
    step_time=dt,
    stepwise_method='simple', #Changes 'mix' to 'simple' so calculation uses single
propagation scheme only
    ps_method=out, # 'ps1' first, 'vmf' second
    rank=20, #Constant rank along the simulation
  )
  progress_bar = tqdm(propagator, total=ceil(end_time / dt))
  for _t in (progress_bar):
    progress_bar.set_description(f'@{_t:.2f} fs')
```

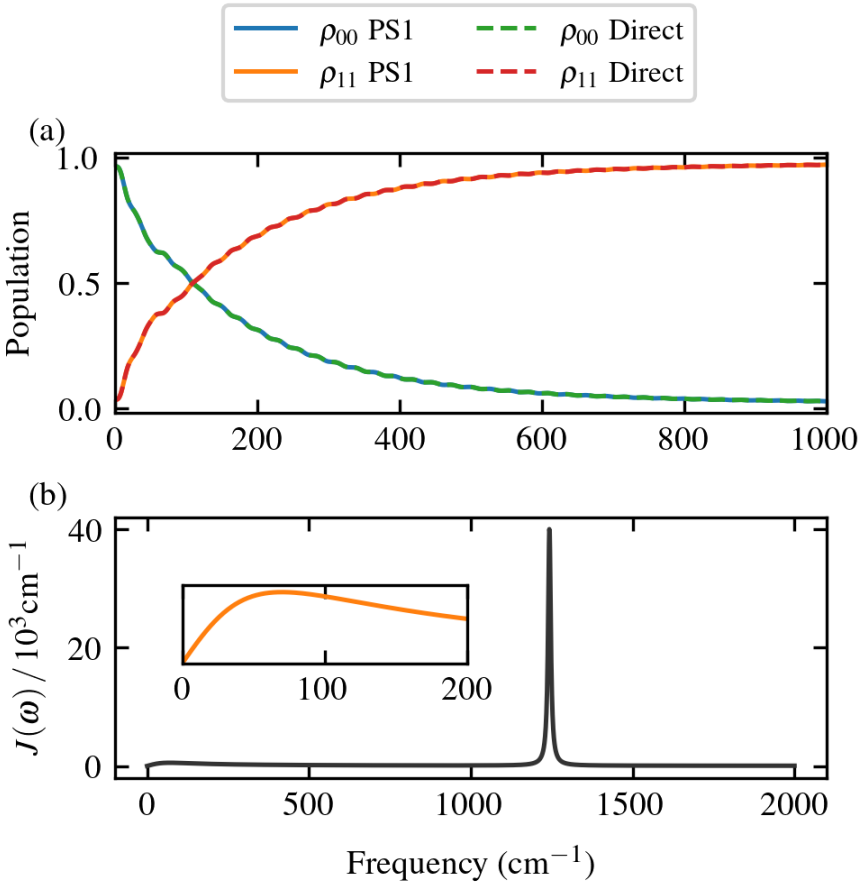
Exercise 5: Different Propagation Methods

- **Do the methods agree?** run `example5.py, example5_plot.py.` Plot the **ps1** and **vmf** dynamics and the difference between them. Note the iterations per second. Was one method faster?

- **A third scheme:** add `'ps2'` to `ps_methods`. Ps2 is like ps1 but **adaptively changes** the tensor ranks. Compare the results. No need to wait until the run fully completes.

- **Default strategy:** drop `stepwise_method` & `ps_method` to get TENSO's default `'mix'`: adaptive ps2 executed initially and then vmf executed at later times. How does this compare?

Results



There should be no numerical difference between propagation schemes when converged.

Ps1 is usually fastest while ps2 will usually be very slow in comparison to the others.

The default mixed propagation scheme is often very fast and robust.

In this case ps1 is faster than the default as the required rank is small.

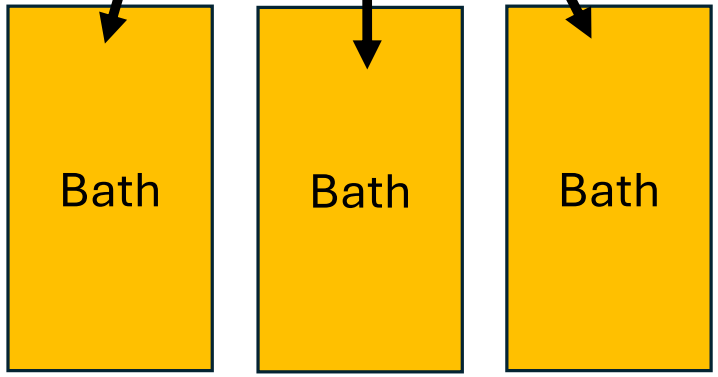
Bigger Example: Fenna-Matthews-Olson Complex

The FMO is a piece of photosynthetic machinery in bacteria and a **common quantum dynamics benchmark**

- The FMO complex experiences a **highly structured SD** but is generally **treated with DL** due to expense
- We compare population dynamics **with DL and two structured spectral densities**
- A separate bath is coupled to each site in the system

$$H_{\mathrm{S}} = \sum_i \epsilon_i |i\rangle\langle i| + \sum_{i \neq j} V_{ij} \left(|i\rangle\langle j| + |j\rangle\langle i|\right)$$

$$H_{\mathrm{S}} = \begin{bmatrix} 200 & -87.7 & 5.5 \\ -87.7 & 320 & 30.8 \\ 5.5 & 30.8 & 0 \end{bmatrix}$$

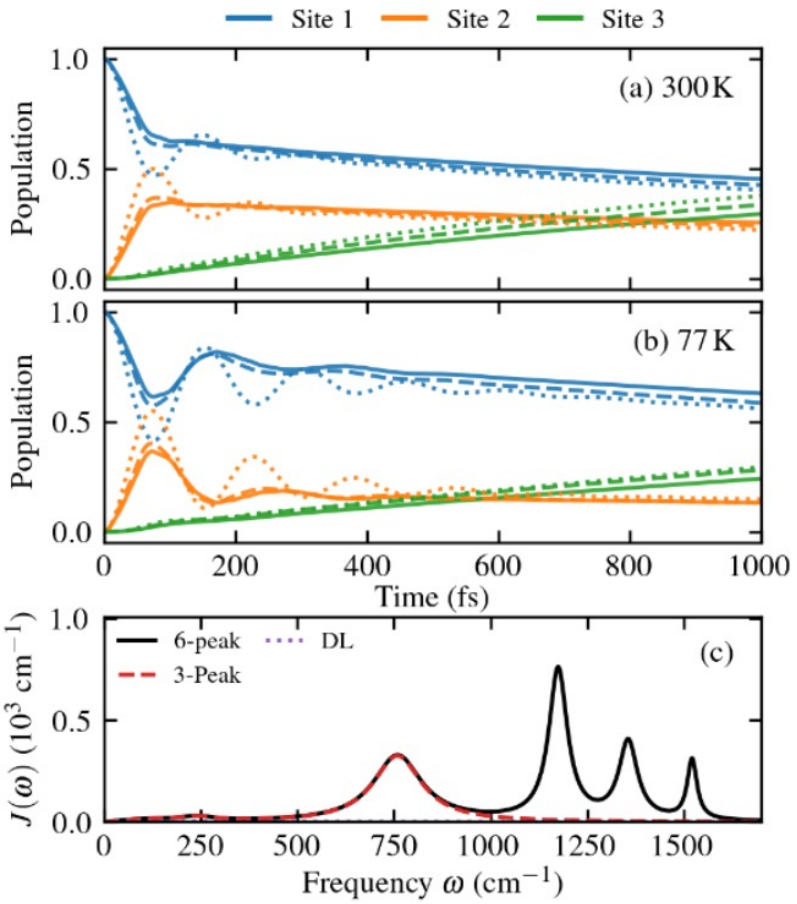


Bigger Example: Fenna-Matthews-Olson Complex

The FMO is a piece of photosynthetic machinery in bacteria and a **common quantum dynamics benchmark**

- The most expensive calculation here requires **45 features**
- Structured spectral density dynamics **differ considerably from the DL** case

Do this at home or tonight!



Convergence in TTN HEOM

Example: Spin-boson with a narrow Brownian oscillator.

Low-temperature corrections (`n_ltc`, default 0): too few give spurious oscillations and unphysical results

Time step (`step_time`): should be ≤ 1/20 of the fastest timescale

Hierarchy depth (`dim`, default 5): insufficient depth produces spurious recurrences

Tensor rank: insufficient rank results in instabilities

Convergence in TTN HEOM

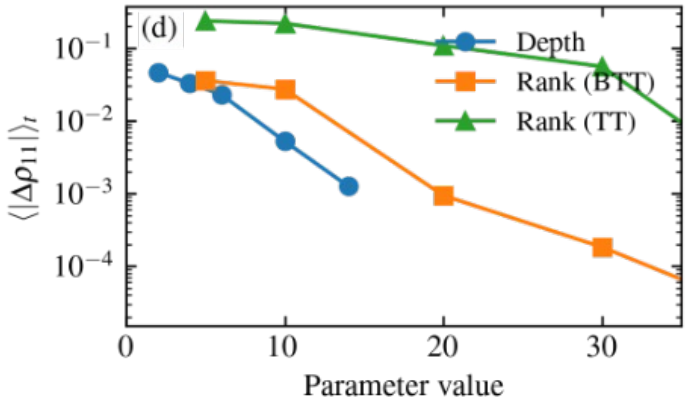
Example: Spin-boson with a narrow Brownian oscillator.

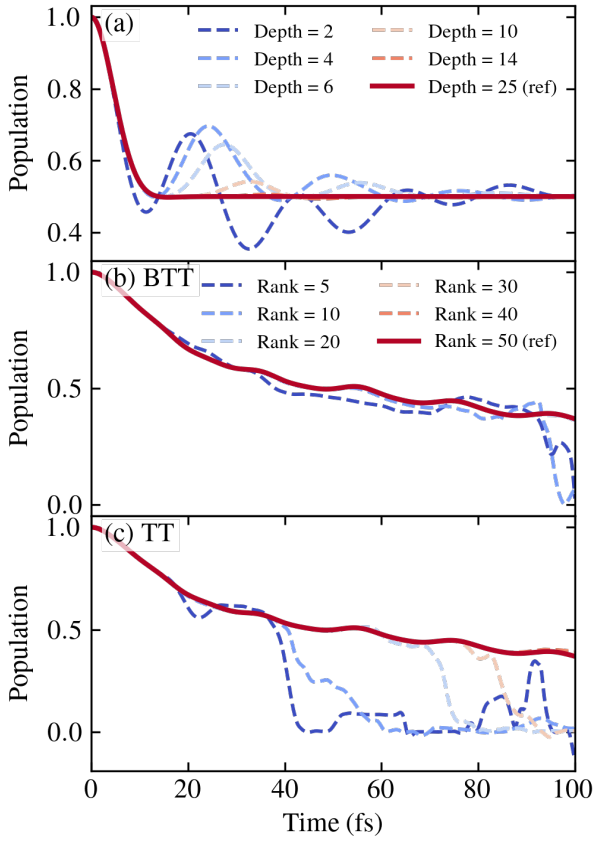
Low-temperature corrections (`n_ltc`, default 0): too few give spurious oscillations and unphysical results

Time step (`step_time`): should be ≤ 1/20 of the fastest timescale

Hierarchy depth (`dim`, default 5): insufficient depth produces spurious recurrences

Tensor rank: insufficient rank results in instabilities





Advanced Features: Custom Correlation Functions

There are alternate ways to find correlation functions

Correlation functions **must be of the form:**

$$C(t) = \sum_{k=1}^{K} c_k e^{\gamma_k t} \qquad C^*(t) = \sum_{k=1}^{K} \bar{c}_k e^{\gamma_k t}$$

Can be input manually into TENSO

Units: the c_k have units of energy squared; γ_k of energy

If there is a **complex γ,** the function will introduce the complex conjugate

Real parts of γ must be negative

Unreasonable C(t) will fail

```
from tenso.bath.correlation import Correlation
bath_simulation = Correlation() # Initialize an empty
    correlation object
# Replace contents of these lists with all the complex
     coefficients of the custom correlation function
c_coefficients = [c_1, c_2, c_3]
gamma_coefficients = [g_1, g_2, g_3]
# unit_convert=True indicates coefficients should be
    converted to internal TENSO units
bath_simulation.manual_corr_setup(c_coefficients,
    gamma_coefficients,unit_convert=True)
```

Changing Units in TENSO

Default: **energies (cm^{-1})** **time (fs)** **temperature (K)**

Can be modified in `src/tenso/prototypes/default_parameters.py`

Alter the values in `default_units`

Options for energy: Joules (`"J"`) electron volts (`"eV"`) milielectron volts (`"meV"`)

Options for time: picoseconds (`"ps"`) seconds (`"s"`)

No other options available for temperature

Exercise: Switching units in an example to meV (1 cm^{-1} = 0.12398 meV).

Note: the system-bath coupling operator is unitless (units are carried by the reorganization energy).

Troubleshooting

1. **Convergence**: depth, rank, low temperature corrections and time step must be appropriate

- Also error tolerances for the numerical integrators of vmf and the maximum rank allowed by ps2 may need adjustment

- Issues detected by convergence testing

- Parameters are tightened until agreement is reached

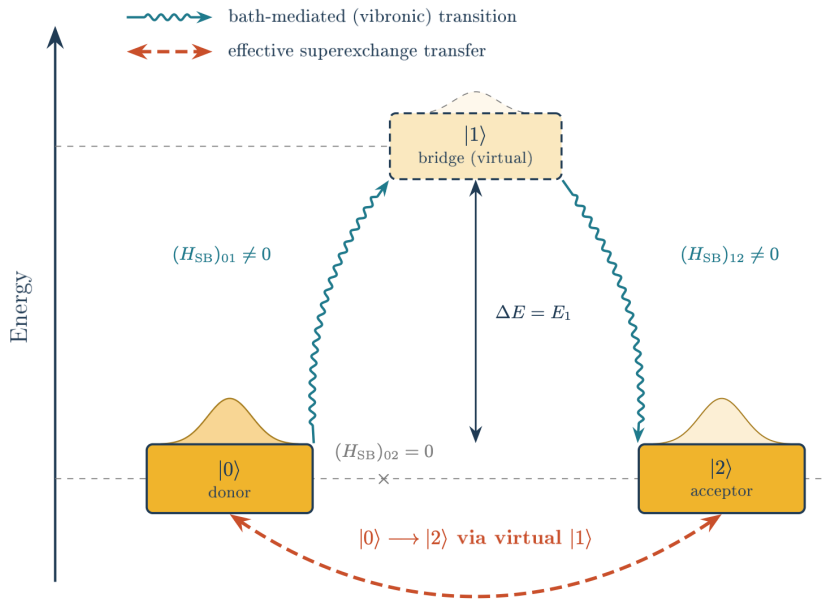
Troubleshooting

2. **Numerical problems**: ps1 and ps2 and even vmf become unstable if the time step is too large

- Usually, the simulation will fail to complete or lead to unphysical norm
- Decreasing the time step or changing the propagation method to fix the problem
- Calculations with strong correlations between all degrees of freedom may not be compatible with tensor network compression

Capstone Project: Superexchange

A three-level system with two degenerate states



$$H = \begin{bmatrix} 0 & 0 & 0 \\ 0 & E_1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$H_{\mathrm{SB}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Population moves from state 0 to state 2 via system bath coupling without accumulating in state 1

Ubiquitous mechanism in electron transfer and molecular electronics

Capstone Project: Superexchange

$$H = \begin{bmatrix} 0 & 0 & 0 \\ 0 & E_1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad H_{\mathrm{SB}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Set up this problem with E_1=1240, and the Brownian term from the spin-boson bath.

Do you observe superexchange?

Change to a two identical bath setup with: $H_{\mathrm{SB}_1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ $H_{\mathrm{SB}_2} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

How does this impact dynamics?

Capstone Project: Superexchange

We will split into groups to investigate.

Group 1: How does temperature impact dynamics?

Group 2: How does the width of the Brownian oscillator feature impact dynamics?

Group 3: How does the frequency of the Brownian oscillator feature impact dynamics?

Work together but run your own computations.

Capstone Project 2: Entanglement Sudden Death

- **Entanglement** is an important quantum resource. For qubits it is **measured by concurrence** determined from the eigenvalues of *R*

$$C(\rho) = \max(0, \lambda_1 - \lambda_2 - \lambda_3 - \lambda_4) \qquad R = \sqrt{\sqrt{\rho}\tilde{\rho}\sqrt{\rho}} \qquad \tilde{\rho} = (\sigma_y \otimes \sigma_y)\rho^*(\sigma_y \otimes \sigma_y)$$

- Set up a system with **two identical, independent TLSs** with H = $10\sigma_z$. Couple each to an independent bath using σ_x.

$$H_{\mathrm{S}} = \frac{\hbar\omega_1}{2}{\sigma_z}^{(1)} + \frac{\hbar\omega_2}{2}{\sigma_z}^{(2)} \qquad H_{\mathrm{SB}} = \hbar\sigma_x^{(1)} \otimes X_{\mathrm{B}}^{(1)} + \hbar\sigma_x^{(2)} \otimes X_{\mathrm{B}}^{(2)},$$

- Initialize the spins in a **maximally entangled (Bell) state** and run dynamics for at least 150 fs

- Write a script or use the one provided (next slide) to **plot the concurrence** between the two spins over time

- What happens to the concurrence as opposed to the coherence between the two spins? How does **changing your bath parameters impact the decay of the coherence or concurrence**?

Capstone Project 2: Entanglement Sudden Death

- **Entanglement** between two qubits is an important quantum resource, often **measured by concurrence** determined from the eigenvalues of *R*

$$C(\rho) = \max(0, \lambda_1 - \lambda_2 - \lambda_3 - \lambda_4) \qquad R = \sqrt{\sqrt{\rho}\tilde{\rho}\sqrt{\rho}} \qquad \tilde{\rho} = (\sigma_y \otimes \sigma_y)\rho^*(\sigma_y \otimes \sigma_y)$$

These functions calculate concurrence for a single four by four matrix or a series of them:

```python
def concurrence_wotters(rho4):
    sy = np.array([[0, -1j], [1j, 0]]); Y = np.kron(sy, sy)
    R = rho4 @ Y @ rho4.conjugate() @ Y
    evals = np.linalg.eigvals(R)
    evals = np.real(np.clip(evals, 0, None))
    lam = np.sort(np.sqrt(evals))[::-1]
    return float(max(0.0, lam[0] - lam[1] - lam[2] - lam[3]))

def concurrence_series(rho):
    if rho.shape[1:] != (4, 4): return None
    C = np.empty(rho.shape[0])
    for i in range(rho.shape[0]): C[i] = concurrence_wotters(rho[i])
    return np.clip(C, 0, None)
```

Controlling Parallelization

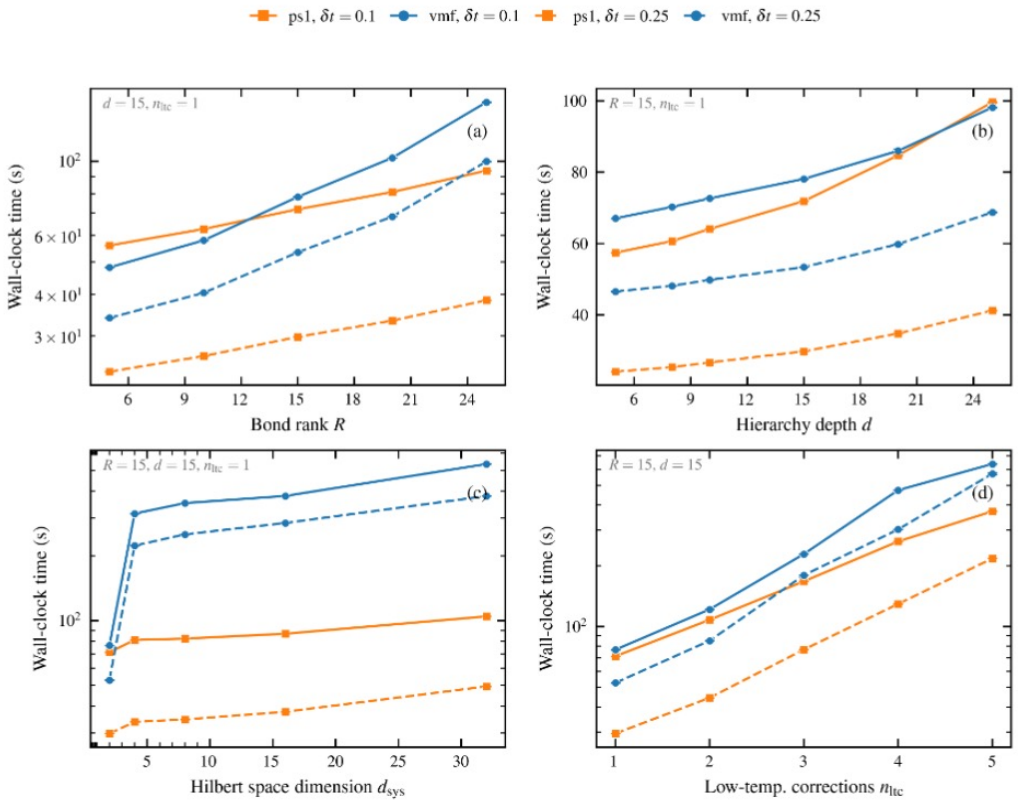
- Pytorch partially implements parallelization
- By default, pytorch will start as many parallel threads as possible
- In TENSO, benefits from parallelization typically cap at 2-4 threads
- Pytorch thread number can be controlled adding _opt.set_num_threads(n) in the file /libs/backend.py

```
import torchdiffeq
try:
    from scikits.odes.odeint import odeint as sundials_odeint
except ImportError:
    sundials_odeint = None

# N_PROCESS = mp.cpu_count()
# import opt_einsum as oe
MAX_EINSUM_AXES = 52  # restrition from torch.einsum as of PyTorch 1.10
PI = math.pi

# PyTorch package settings
_opt.set_grad_enabled(False)  # disable autograd by default
_opt.set_num_threads(2)
```

TENSO: Performance Metrics



Performance depends heavily on the number of bath features

Rank, depth, and system size also impact run time.

System characteristics, i.e. whether the equations of motion are stiff, can have a strong impact as well

Advanced Features

- Define your own HEOM variant by varying metric and bexcitonic representation
- Define arbitrary order for tensors and arbitrary tensor networks
- Run MCTDH with TEDOPA bath discretization
- Implement another master equation whose generator is in a sum of products form

¡Se acabó!

The end